УДК 004.657

ИНДЕКСИРОВАНИЕ ПРОСТРАНСТВЕННЫХ ДАННЫХ В СУБД MICROSOFT SQL SERVER 2000

Н.А. Шестаков

Томский политехнический университет
E-mail: ShestakovNA@ce.cctpu.edu.ru

В среде СУБД Microsoft SQL Server 2000 реализованы 2 схемы пространственного индексирования. Проведено экспериментальное исследование реализованных методов для оконных запросов. Выполнено сравнение реализованных методов с имеющимися в данной СУБД стандартными средствами индексирования. Для нахождения квадрантного разбиения в методах Z- и XZ-индексирования предложен эвристический алгоритм, который дает меньшую ошибку аппроксимации по сравнению со стандартным алгоритмом.

Введение

Развитие геоинформационных систем (ГИС) в последние годы привело к повышенному вниманию со стороны разработчиков систем управления базами данных (СУБД) к работе с пространственными данными. Возрастающие требования ГИС к объемам и надежности хранения данных привели к их интеграции с мощными универсальными СУБД, как правило, независимых разработчиков. От СУБД в данном случае требуется поддержка работы с пространственными данными (поддержка пространственных типов и пространственное индексирование).

СУБД MS SQL Server является одним из лидеров на рынке серверных СУБД. Несмотря на то, что большинство ее основных конкурентов уже имеют хотя бы базовые средства для хранения пространственных данных, в продуктах от Microsoft

они отсутствуют (как в MS SQL Server 2000, так и в 2005 [1, 2]). В данной работе рассматривается вариант реализации пространственного индексирования в СУБД MS SQL Server 2000 методами Z- и XZ-индексирования, которые сравниваются с работой стандартных индексов. Численный эксперимент дает представление, насколько можно повысить скорость выполнения оконных запросов в различных условиях.

1. Пространственные СУБД и пространственное индексирование

Основные особенности пространственных СУБД заключаются в том, что они поддерживают соответствующие типы данных, запросы в языке и механизмы индексирования.

В качестве примеров пространственных запросов можно назвать [3, 4]:

- оконные запросы (*window query* или *range query*);
- поиск *k* ближайших соседей (*k nearest neighbors query* или *k-NN query*);
- запросы с пространственным соединением (*spatial join*).

В данной работе исследуется эффективность выполнения оконных запросов.

В схеме работы пространственных запросов обычно выделяют две стадии или две *ступени фильтрации*. СУБД, обладающие слабой пространственной поддержкой, отрабатывают только первую ступень (грубая фильтрация). Как правило, на этой стадии используется приближенное, аппроксимированное представление объектов. Самый распространенный тип аппроксимации – минимальный ограничивающий прямоугольник (MBR – *Minimum Bounding Rectangle*) [5]. Функция вторичной фильтрации возлагается на ГИС или другое клиентское приложение. Работая с такой СУБД, как MS SQL Server, вполне оправдано использовать представление объектов в виде MBR на уровне базы данных.

Решающим фактором при оптимизации запросов к БД является правильное использование индексов. Для пространственных типов данных существуют особые методы индексирования, например, на основе *R*-деревьев [6, 7].

Подробный обзор методов пространственного доступа (*spatial access methods*) присутствует в [3]. Если СУБД не обладает средствами пространственного индексирования, то возможно реализовать такое индексирование своими силами. Эти расширения иногда называют «надстройками», «картриджами» или «плагинами». Реализуются они при помощи триггеров и хранимых процедур на языке, используемом в СУБД, либо же внешних процедур, или средствами сервера приложений (*application server*), если встроенных языковых средств недостаточно.

Метод Z-индексирования. Метод основан на использовании кривой покрытия, или кривой, заполняющей пространство (*кривая Пеано, space filling curve*). Кривая покрытия разбивает пространство на области (ячейки), каждой из которых присваивается номер (*код Пеано*) и, таким образом, задается порядок. Протяженные объекты и область запроса аппроксимируются множеством ячеек (и соответствующих кодов Пеано), пример показан на рис. 1. Каждой ячейке соответствует диапазон возможных кодов Пеано, что дает возможность превратить запрос по области в диапазонный запрос, который оптимизируется при помощи стандартного одномерного индекса.

Каждому значению кода Пеано можно поставить в соответствие узел квадродерева. Поэтому, пользуясь терминологией квадродеревьев, упорядочиваемые ячейки будем называть квадрантами. *Квадрант* – это область пространства, полученная рекурсивным делением плоскости на 4 равные части.

XZ-индексирование. Недостатком Z-индексирования при работе с неточечными (протяженными) объектами является то, что одному объекту сопоставляется несколько чисел Пеано, т.е., несколько индексов. Наличие отношения «один ко многим» приводит к тому, что индексы хранятся отдельно от индексируемой таблицы. Это приводит к возникновению дополнительного соединения на таблицу с индексами при выполнении запроса, и данные в основной таблице нельзя кластеризовать по Z-индексу.

Метод XZ-индексирования, описанный в [8], модифицирует классический метод таким образом, что одному объекту соответствует только одно значение Пеано кода. Однако ошибка аппроксимации в таком методе выше.

Квадрантное разбиение области. При индексировании объектов методом Z-индексирования, необходимо представить объект в виде множества квадрантов, которые помещаются в БД в виде кодов Пеано. Пример разбиения прямоугольной области на квадранты показан на рис. 1. Число квадрантов зависит от разрешения базовой сетки (размера минимального квадранта, определяемого максимальной глубиной квадродерева) и размера разбиваемой области.

При выполнении оконного запроса, область запроса также разбивается на квадранты (для методов Z- и XZ-индексирования). Каждому квадранту соответствует диапазон кодов Пеано. Запрос представляет из себя выборку всех объектов, чьи коды Пеано попадают в эти диапазоны.

Точность представления объекта будет тем лучше, чем больше число квадрантов. Количественно ее можно оценить ошибкой аппроксимации. Под ошибкой аппроксимации σ будем понимать отношение суммарной площади квадрантов ΣS_q к площади разбиваемой фигуры S минус единица.

$$\sigma=\Sigma S_q/S-1=(\Sigma S_q-S)/S.$$

Как правило, существует некоторое разумное ограничение на число квадрантов. При построении индексов объектов оно составляет единицы, при

построении диапазонов в оконном запросе – сотни. При этом возникает задача получить такое разбиение, которое давало бы наименьшую ошибку при заданном ограничении на число квадрантов. Такое разбиение будем считать оптимальным.

Алгоритм разбиения на квадранты. В [8] описан алгоритм рекурсивного разбиения с ограничением глубины рекурсии. Он не дает оптимального разбиения, но, по крайней мере, лучше, чем просто разбивать до достижения максимальной глубины квадродерева. Идея алгоритма состоит в том, чтобы рекурсивно разбивать пространство до достижения некоторой определенной глубины рекурсии, которая рассчитывается априори, исходя из ограничения на допустимое число квадрантов N_{max}.

Поскольку ограничение глубины рекурсии позволяет лишь грубо приблизиться к границе числа квадрантов N_{max} при разбиении, то результат работы алгоритма обычно не оптимален.

Автором был предложен алгоритм, в котором процесс разбиения регулируется некоторой эвристикой, которая определяет, какой квадрант в текущем разбиении нужно дробить в следующую очередь. Он тоже не находит оптимальное разбиение, но результат достаточно близок к оптимальному.

Входными параметрами алгоритма являются: N_{max} – максимально допустимое число квадрантов в разбиении; *Rect* – прямоугольная область, которую необходимо разбить на квадранты.

Выходным параметром алгоритма является *qlist* – список квадрантов.

Ниже приведено пошаговое описание его работы.

Шаг 1. Для квадранта *q* установить максимальный размер, равный размеру всего пространства.

Шаг 2. Поместить *q* в список *qlist.*

Шаг 3. Если длина списка *qlist* равна N_{max}, то завершить алгоритм.

Шаг 4. Если в списке *qlist* нет квадрантов, которые можно разбить без переполнения списка *qlist*, то завершить алгоритм.

Шаг 5. Выбрать из списка *qlist* квадрант *q* с максимальным значением эвристики.

Шаг 6. Разбить *q* на подквадранты и поместить полученные подквадранты, которые пересекают область *Rect*, в список qlist.

Шаг 7. Перейти на *Шаг 3.*

Алгоритм является жадным, т.к. каждая итерация улучшает текущее разбиение и приближает его к итоговому результату.

Смысл подсчитываемой эвристики – площадь уменьшаемого пространства при разбиении квадранта, отнесенная к числу квадрантов, на которое увеличится разбиение.

Дело в том, что не каждое разбиение приводит к уменьшению площади аппроксимации. Например, разбиение квадранта 1 (рис. 1) на 4 подквадранта не приведёт к улучшению текущего разбиения, так как каждый из этих подквадрантов все ещё будет пересекать исходную область. Однако при дальнейшем разбиении уже будут появляться пустые квадранты, которые будут удалены из списка и уменьшат ошибку аппроксимации. Квадрант 2 при разбиении сразу даст уменьшение ошибки, хотя площадь пустого пространства в нем меньше, чем у квадранта 1. Первое разбиение квадранта 3 также ведет к уменьшению площади аппроксимации, хотя это уменьшение невелико. Квадрант 1 потенциально более выгоден для разбиения, чем квадрант 3, хотя и разбиений потребуется больше. Нужно только проверить, возможно ли такое разбиение с учетом ограничения N_{max}.

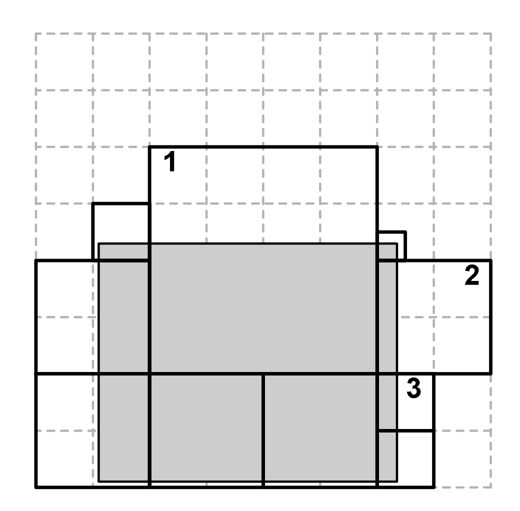


Рис. 1. *Оптимальное разбиение, N_{max}=10*

Поэтому для каждого квадранта подсчитывается число подквадрантов N_{suc}, на которое нужно разбить исходный квадрант до достижения первого «удачного» разбиения (приводящего к уменьшению ошибки). Также подсчитывается само уменьшение dS (в абсолютных единицах площади). Очередной квадрант для разбиения выбирается с максимальным значением эвристики $E=dS/(N_{suc}-1)$. Но при этом необходимым условием является, чтобы N_{suc} не превышало разницу между текущим количеством квадрантов и максимальным.

Если несколько квадрантов имеют одинаковое значение E, то выбор происходит по максимальной площади пустого пространства квадранта:

$$S_{empty}=S_q-S(q\cap Rect),$$

где S_q – площадь квадранта, $S(q\cap Rect)$ – площадь пересечения квадранта и разбиваемой области.

Было установлено, что для квадрантов, которые пересекаются границей области вертикально (как квадранты 2 и 3 на рис. 1) или горизонтально (как квадрант 1), величина N_{suc} зависит от величины зазора между границей области и внешней границей квадранта (зазор соответствует пустому пространству квадранта). Если нормировать сторону квадранта единицей и принять d – величину зазора, то

$$N_{suc}=2\times2^{lb(d)}-2,$$

где $lb(d)$ – позиция первого единичного бита в двоичном представлении d (например, $lb(0.1101)=1$; $lb(0.001)=3$). При этом

$$dS=S_q/2^{lb}.$$

Для угловых квадрантов в качестве d берется максимальная величина зазора. Это, в принципе, может привести к не совсем точному подсчету эвристики, но не сказывается на качестве работы алгоритма в целом.

Вычислительная сложность эвристической функции невысока, поскольку при расчете используются только операции сложения/вычитания и битовые сдвиги.

Для разных значений N_{max} была подсчитана средняя ошибка аппроксимации σ, получающаяся в результате работы алгоритмов. Величина N_{max} была взята из двух диапазонов: от 4 до 8 (характерные значения при построении Z-индексов объектов) и от 400 до 800 (характерные значения при разбиении области запроса для формирования интервалов Z-значений). Результаты приведены в таблице.

Таблица. *Ошибка аппроксимации σ для рекурсивного и эвристического алгоритмов*

N_{max}	Алгоритм	
	Рекурсивный	Эвристический
4	3,5	2,4
6	2,8	1,2
8	2,7	0,9
400	0,035	0,019
600	0,024	0,012
800	0,015	0,009

Эвристический алгоритм дает меньшую (от 1,5 до 3 раз) ошибку аппроксимации. Эта разница существенна для небольших значений N_{max}. Когда ограничение на число квадрантов достаточно велико, эта разница незначительна, поскольку мало само значение ошибки.

Что касается скорости работы алгоритмов, то здесь преимущество на стороне рекурсивного метода – его время работы асимптотически линейно по отношению к N_{max}. Трудоемкость работы эвристического алгоритма пропорциональна N_{max}^2. Впрочем, для небольших значений N_{max} время работы обоих алгоритмов пренебрежимо мало по сравнению со временем выполнения SQL запроса.

2. Пространственное индексирование в среде MS SQL Server 2000

Проблема выбора модели данных для тестовой системы. Схема данных (объектная и реляционная) определяется спецификой задачи. Что касается непосредственно пространственной части, то перед проектировщиком могут вставать различные вопросы по поводу представления пространственной информации. В данном случае при выборе схемы было решено опираться на стандарт OpenGIS Simple Features Specifications For SQL [9] (далее по тексту стандарт OpenGIS), хотя не ставилось цели полного соответствия стандарту, для создаваемой системы-прототипа это не требуется.

Стандарт OpenGIS определяет различные схемы данных для двух языковых SQL сред: SQL92 и SQL92 with Geometry Types. Поскольку последняя предполагает наличие геометрических типов в самом языке, то для MS SQL Server она не подходит. Для SQL92 определено два возможных варианта реализации: с хранением элементов геометрических фигур и с использованием так называемых «больших двоичных объектов» (BLOB) для хранения геометрии. Автором был выбран второй способ, так как он не ограничивает геометрическое описание фигур примитивами, определенными стандартом и, по некоторым наблюдениям, чаще используется.

Особенности реализации в среде MS SQL Server 2000. Геометрические объекты представлены в виде прямоугольников (MBR), которые описываются в таблице набором четырех атрибутов: x_0, x_1, y_0, y_1. Для хранения дополнительной информация о геометрии объектов предусмотрен атрибут *data*, имеющий тип *image*, который является BLOB типом в реализации MS SQL Server.

Для метода XZ-индексирования основная таблица расширена дополнительным атрибутом – значением XZ-индекса (тип *int* – 4 байта). В методе Z-индексирования индексы хранятся в отдельной таблице, связанной с основной по внешнему ключу.

Никакой атрибутивной информации об объектах в экспериментальной БД не хранилось. Предполагается, что пространственная и атрибутивная информация разделены по разным таблицам, а здесь нас интересует только пространственная составляющая.

Метод независимых индексов. Данный метод не использует пространственного индексирования. Смысл его использования заключается в оценке того, насколько стандартные средства могут быть хуже или лучше тех, которые придумываются взамен.

В основной таблице по координатам MBR построены некластерные индексы (независимо по каждому столбцу). Запрос в данном случае представляет простую выборку на пересечение четырех диапазонов. Простота этого способа очевидна. Во-первых, не требуется строить свои индексы, во-вторых, данные выбираются простейшим запросом.

Метод Z-индексирования. Входной информацией для алгоритма являются координаты окна запроса. На выходе формируется множество объектов, соответствующих исходному окну. Последовательность действий такая:

Шаг 1. Преобразовать окно запроса в набор интервалов Z-значений:

1.1. Разбить окно на квадранты.

1.2. Каждый квадрант преобразовать в соответствующий интервал.

1.3. Выполнить слияние близких интервалов из полученного набора.

Шаг 2. По набору интервалов сформировать и выполнить SQL-запрос.

Шаг 3. Дополнительно отфильтровать полученное множество объектов, используя информацию об MBR объектов.

На **шаге 2** происходит выборка идентификаторов объектов из таблицы, хранящей Z-индексы (упорядоченные кластерным индексом), после чего выполняется соединение этого списка на основную таблицу с объектами.

Метод XZ-индексирования. Общий алгоритм совпадает с тем, что используется в методе Z-индексирования, только модифицируется алгоритм построения Z-значений и интервалов. Также на **шаге 2** в запросе не выполняется соединение, поскольку XZ-индексы хранятся в основной таблице, и по полю XZ-индекса построен кластерный индекс.

3. Численный эксперимент

Условия проведения эксперимента. Для проведения экспериментов в среде разработки Borland Delphi 7 было написано тестовое клиентское приложение. Серверная часть работала под управлением СУБД MS SQL Server 2000 (MSDE), запущенной на однопроцессорном компьютере (AthlonXP 2400) с 512 Мб памяти и IDE жестким диском объемом 80 Гб и 8 Мб кэшем. В качестве основных критериев эффективности работы оконных запросов использовались характеристики:

- время выполнения запроса (*execution time*);
- количество обращений к диску (*disk accesses*).

Замеры времени производились на клиентской стороне. Для определения количества дисковых операций измерялись значения глобальных переменных @@TOTAL_READ и @@TOTAL_WRITE.

Эксперимент проводился на 9 наборах искусственно сгенерированных данных, распределенных равномерно по двумерной квадратной области. В наборах варьировалось количество объектов: 200 тыс., 600 тыс., 1 млн; размер объектов: точечный, нормальный и большой. Возможные комбинации количества объектов и размера дают 9 вариантов тестовых наборов данных.

На каждом наборе данных проводилось измерение времени и количества дисковых операций для оконного запроса с использованием каждого из трех реализованных методов. Окно запроса принимало значения 0,01, 0,04, 0,2, 1,0 и 5,0 % от площади всего пространства (все пространство определяется возможным диапазоном координат, в данном случае, от – 32768 до 32767 для используемого типа *shortint*, то есть сеткой 65536×65536). Объекты распределены в пространстве равномерно. Таким образом, размер окна фактически характеризует конечную селективность запроса.

Все значения критериальных параметров получены усреднением по 25 измерениям. Стандартное отклонение от среднего значения в серии составило 10...15 % для значений времени и менее 5 % для числа дисковых операций.

Результаты эксперимента и их обсуждение. На рис. 2 приведены результаты эксперимента для объектов нормального размера. Слева (рис. 2, *а*, *б*, *в*) изображены графики для времени выполнения запросов, справа (рис. 2, *г*, *д*, *е*) – для числа дисковых операций.

По результатам эксперимента можно сделать следующие замечания:

1. Очевидно, что с увеличением объема базы данных время работы всех запросов увеличивается. При этом время в методе независимых индексов и в методе Z-индексирования растет быстрее, чем в методе XZ-индексирования.
2. Эффективность запроса в методе независимых индексов не зависит от размера окна запроса (в исследуемом диапазоне). Методы Z- и XZ-индексирования гораздо чувствительнее к размеру окна.
3. Можно заметить не совсем понятное на первый взгляд поведение метода Z-индексирования, когда число обращений к диску для определенного значения размера окна слишком высокое (как на рис. 2, *г* и *д*) для значения размера окна 0,04 %). Причина такого скачка в том, что при соединении таблиц оптимизатор выбирает неудачный план запроса. Если же непосредственно указывать оптимизатору, какой тип плана использовать, это приводит к снижению эффективности соединения для малых размеров окна.
4. Размер хранимых объектов не оказывает заметного влияния на время выполнения запросов (поэтому приведены результаты только для объектов нормального размера). Исключение – метод Z-индексирования, который для точечных объектов работает немного быстрее. Это объясняется тем, что точечные объекты не разбиваются при построении индекса, и таблица с Z-индексами получается меньшего размера.
5. При одинаковых значениях дисковых операций, время работы метода независимых индексов больше, чем у других методов. Видимо, это говорит о том, что этот метод использует больше ресурсов процессора, чем другие.

Выводы по эксперименту. Можно сказать, что при определенных условиях (достаточно хорошей селективности и достаточно большом количестве объектов) использование специальных приемов для индексирования пространственных данных может дать достаточно ощутимый прирост производительности, если использовать метод XZ-индексирования. Метод Z-индексирования показал себя неудовлетворительно даже по сравнению с использованием независимых индексов (за исключением случая, когда селективность запроса очень небольшая – 0,01 %). В целом, для повышения ско-

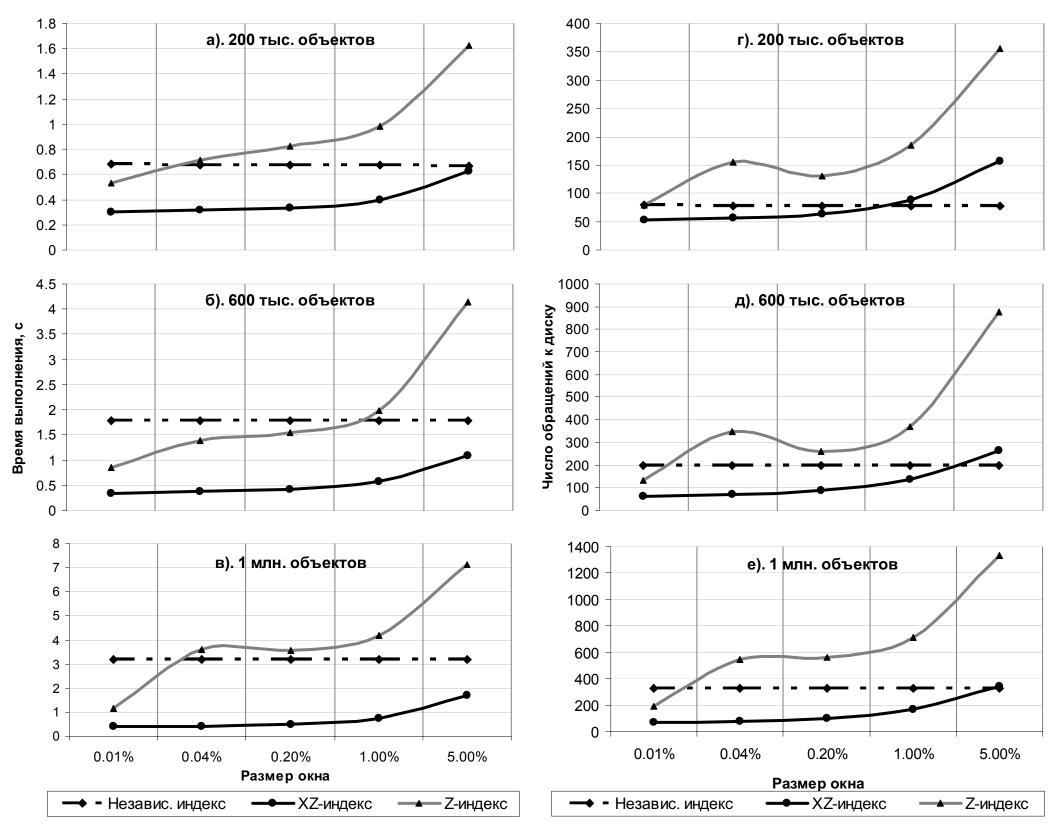


Рис. 2. *Результаты эксперимента (слева – замеры времени, справа – замеры числа дисковых операций)*

рости выполнения оконных запросов, можно рекомендовать использовать XZ-индексирование для таблиц, в которых содержится более 600 тыс. записей при выборке по окну размером менее 1 %.

Заключение

Для тестовой базы данных, содержащей пространственные объекты и работающей под управлением СУБД MS SQL Server 2000, были реализованы методы пространственного индексирования (Z-индексирование и XZ-индексирование). Было проведено тестирование производительности индексирования по сравнению со стандартными имеющимися в СУБД средствами (метод независимых индексов). Результаты показали, что метод XZ-индексирования может существенно ускорить работу оконных запросов при определенных условиях (например, для селективности запроса менее 1 % и количестве объектов более 600 тыс. – более чем в 3 раза). Однако при достаточно большом размере окна запроса или на маленьких объемах данных метод XZ-индексирования может работать даже медленнее, чем стандартный.

Предложенный эвристический алгоритм квадрантного разбиения обеспечивает меньшую ошибку аппроксимации, чем стандартный алгоритм при том же заданном ограничении на число квадрантов в разбиении.

СПИСОК ЛИТЕРАТУРЫ

1. Francica J. A Spatial Database Technology Update with Dr. Ignacio Guerrero, Intergraph (interview) // Directions Magazine. – 2003. – № 1(Jan), http://www.directionsmag.com/
2. Microsoft SQL Server: Future Plans for Supporting Spatial Data // Directions Magazine. – 2003. – № 10. http://www.directionsmag.com/
3. Gaede V., Gunter O. Multidimensional Access Methods // ACM Comput. Surv. – 1998. – V. 30. – № 2 (June). – P. 170–231.
4. Guting R.H. An Introduction to Spatial Database Systems // VLDB Journal. – 1994. – V. 3. – № 4. – P. 297–308.
5. Papadias D., Theodoridis T. Spatial Relations, Minimum Bounding Rectangles and Spatial Data Structures // Technical Report KDB-SLAB-TR-94-04, http://www.cs.ust.hk/faculty/dimitris/PAPERS/ijgis97.pdf
6. Guttman A. R-trees: A dynamic index structure for spatial searching // Proc. of the ACM SIGMOD Intern. Conf. on Management of Data, 1984. – P. 47–54.
7. Faloutsos C., Kamel I. Hilbert R-Tree: An Improved R-Tree Using Fractals // Department of CS, University of Maryland, Technical Research Report TR-93-19, https://drum.umd.edu/dspace/handle/1903/581
8. Bohm C., Klump G., Kriegel H.-P. XZ-Ordering: A Space-Filling Curve for Objects with Spatial Extension // University of Munich, Computer Science Institute, http://www.dbs.informatik.uni-muenchen.de/Publikationen/Papers/SSD-XZ-Order.final.pdf
9. OpenGIS Simple Features Specification For SQL Rev. 1.1 // Open GIS Consortium, 1999. http://portal.opengeospatial.org/files/?artifact_id=829